Technical Note

1975-18

J. H. McClellan

Hardware for the Fermat Number Transform

1 April 1975

Prepared for the Ballistic Missile Defense Program Office,
Department of the Army,
under Electronic Systems Division Contract F19628-73-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This program is sponsored by the Ballistic Missile Defense Program Office, Department of the Army; it is supported by the Ballistic Missile Defense Advanced Technology Center under Air Force Contract F19628-73-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Eugene C. Raabe

Eugene C. Raabe, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

HARDWARE FOR THE FERMAT NUMBER TRANSFORM

J. H. McCLELLAN
Group 24

TECHNICAL NOTE 1975-18

1 APRIL 1975

LEXINGTON MASSACHUSETTS

ABSTRACT

The design and implementation of a hardware Fermat Number Transform (FNT) is described. The arithmetic logic design is treated in detail and a new data representation for integers modulo a Fermat number is derived. Some results of filter implementation with the FNT are shown to illustrate the use of the hardware. Finally, the FNT is compared with the Fast Fourier Transform (FFT) on the basis of hardware required for a pipeline convolver.

TABLE OF CONTENTS

I. Introduction

The use of number theory transforms for implementing digital convolution is attractive from a theoretical point of view because it is possible to derive a transform that requires no multiplications. Since multipliers are a major hardware expense in Fast Fourier Transform (FFT) pipeline convolvers or in direct form convolvers, the potential for building cheaper and/or faster convolvers should lie with number theory transforms. Many other factors cloud the picture and the savings in multiplier hardware can be offset by increased memory hardware and transform size in some cases. It is the purpose of this paper to examine some of these hardware issues.

In the realm of radar signal processing, the potential for greater speed is worth exploring. For this reason a small prototype number theory transform has been constructed. This hardware consists of the computational element (butterfly)for a 64-point, 16-bit Fermat Number Transform (FNT).

In the process of designing the butterfly, a new coding scheme for the data was developed to facilitate the arithmetic operations modulo the Fermat number. The experience gained in designing and building this hardware is the basis for estimates of the size of pipeline FNT convolvers for possible use in radar signal processors. The result of the hardware sizing of the FNT versus a pipeline FFT indicates that the anticipated savings can be realized for small systems (e.g., length 32 convolution) when the data to be filtered is real. However, in larger systems where one must use two-dimensional convolution to implement the one-dimensional convolution, the savings in multiplier hardware are offset by the increased transform size and the corresponding increase in memory size and reference spectrum multiplier hardware. In this case, when the

data to be filtered are real, the FNT still offers a small decrease in the amount of hardware versus the FFT, but when the data are complex the amount of hardware are much greater than a pipeline FFT.

Finally, some examples of FIR filter implementations using the FNT hardware will be given, in order to illustrate the effects of precision with this approach.

II. Theory

There is a large class of transforms which can be derived when the underlying algebraic structure is assumed to be a finite field or ring (Ref. [1]). When considering hardware implementations, however, only the Fermat number transforms offer the dual advantages of no multiplications in the transform and decomposition into a fast algorithm analogous to the FFT.

The FNT of $\{x(k)\}$ is defined as

$$X(n) = \sum_{k=0}^{N-1} x(k)\ \alpha^{-<nk>} \bmod F_t \qquad (1)$$

where

$F_t = 2^{2^t} + 1$, the t^{-th} Fermat number

N is a power of 2

and α is an Nth root unity (i.e., $\alpha^N = 1 \bmod F_t$ and $\alpha^m \neq 1$, $1 \leq m < N$)

The notation <nk> means nk modulo N.

The only FNT's considered here are those in which α has a simple binary representation, so that the multiplications implied in equation 1 are easy to implement. It is possible to show that for $N = 2^{t+1}$, $\alpha = 2$ is an Nth root of unity [2] . In this case the multiplications in (1) become bit shifts. Agarwal and Burrus [3] showed that $\alpha = 2^{3 \cdot 2^{t-2}} - 2^{2^{t-2}}$ is an Nth root of unity for $N = 2^{t+2}$. Since $\alpha^2 = 2 \bmod F_t$ this α is usually referred to as the square root of 2 and written $\alpha = \sqrt{2}$. For general F_t ($t > 4$), $N = 2^{t+2}$ is the largest power of 2 for which a transform can be defined.

Since $\sqrt{2}$ has a two-bit representation, multiplication by $\sqrt{2}$ can be implemented with one subtraction. In addition, the case of N equal to a power of two is important because it allows factorization of the transform into a structure similar to the FFT algorithm for the Discrete Fourier Transform. These properties of the FNT are explained in detail in reference 3 .

A fundamental constraint imposed by the transform definition is that the wordlength of the arithmetic (determined by F_t) is linearly related to the transform length. For the case $\alpha = \sqrt{2}$, N = 4 x wordlength. It is possible to ease this constraint by using a two-dimensional implementation of the one-dimensional convolution [4] . With $\alpha = \sqrt{2}$ an N x N two-dimensional transform can be defined with $N = 2^{t+2}$. Using this two-dimensional transform a one-dimensional circular convolution of length $\frac{1}{2}N^2$ can be implemented. A 50% loss in convolutional efficiency is incurred in using a two-dimensional transform, because with a length N^2 one-dimensional FFT, a length N^2 one-dimensional circular convolution can be realized.

Returning to the definition in equation (1), note that the transform is defined in the algebraic system of integers modulo F_t. Thus, the implementation of circular convolution using (1) will result in circular convolution modulo F_t. In particular, if the circular convolution

$$Y_n = \sum_{k=0}^{N-1} x_k \, h_{<n-k>} \tag{2}$$

is computed using FNT's the result will be

$$\hat{Y}_n = \left\{ \sum_{k=0}^{N-1} x_k \, h_{<n-k>} \right\} \bmod F_t \tag{3}$$

It is possible to determine Y_n from $\hat{Y}_n$ if and only if Y_n is known a priori to lie in the set $[\,P,\ P + F_t - 1\,]$. With no prior knowledge of the ranges of $\{x_k\}$ and $\{h_k\}$ a conservative estimate of the number of bits for x_k and h_k can be made.

Let $N = 2^n$ and assume that $|x_k| \leq 2^a$ and $|h_k| \leq 2^b$ for $k = 0, 1, \ldots N-1$. Then Y_n will be in the set $[\,-\frac{F_t-1}{2},\ \frac{F_t-1}{2}\,]$ for all possible sequences $\{x_k\}$ and $\{h_k\}$ if and only if $n + a + b \leq 2^t - 1$. This bound is overly conservative in most cases, but it does represent a least upper bound. Letting $t = 4$, $n + a + b \leq 15$. In this case, if $n = 5$, (i.e., a length 21 convolution) then a and b could both be 5, but 5 bits is probably not enough accuracy for convolution. A more likely case would have $a = b = 10$ so that $n + a + b = 25$, and $t = 5$ (33-bit arithmetic) would be necessary. For most typical filtering applications a 33-bit system (i.e., a modulo $2^{32} + 1$ system) would seem to be most appropriate. The problems with precision in a 17-bit system can be overcome in other ways [5].

A drawback of the fact that the wordlength of the system must be a power of 2 is that it is not possible to tradeoff wordlength versus performance as is commonly done in the realization of digital filters. However, the computation of the convolution using the FNT (or any number theoretic transform) is exact. That is, after the quantization of the input data and the filter coefficients, no additional quantization noise is introduced in the filtering process. Thus, the need for simulations of the filtering process is reduced to determining two wordlengths as opposed to present efforts that involve all the internal precisions of the calculation [6].

A more complete discussion of all these theoretical issues can be found in references 1 through 5. In the following sections our attention will focus on the hardware issues encountered in realizing the FNT.

III. Representation of Numbers Modulo 2^t+1

3.1 Modulo 2^t+1 Arithmetic

In this section a new data coding scheme is described for performing arithmetic modulo 2^t+1. The main result is that modulo 2^t+1 arithmetic can be implemented in a manner that is similar to 1's complement arithmetic (i.e., modulo 2^t-1 arithmetic). A description of the arithmetic operations of the FNT when the data was encoded in 2's complement notation can be found in reference [3]. The arithmetic operations of interest are addition, subtraction and multiplication by a power of 2, because these are the basic operations in the butterfly of the FNT using an FFT-like structure.

Recall that in the ring of integers mod 2^t+1, there are 2^t+1 elements. Thus, t + 1 bits are needed to represent all possible numbers. If a binary coding scheme such as 2's complement were used, then whenever the MSB was one, all the other bits would be zero. This combination would represent the number 2^t. Thus, the t^{th} bit is used only in this one case. A new coding scheme is proposed in which the collection of t+1 bits $[\, b_t\ b_{t-1} \ldots b_0 \,]$ represents the number B in the following way:

1. If $b_t=1$ then $B = 0$
2. If $b_t=0$ then $B = -(-1)^{b_{t-1}} 2^{t-1} - (-1)^{b_{t-2}} 2^{t-2} + \ldots - (-1)^{b_0}$

That is, the jth bit has weight 2^j and sign σ_j, where

$$\sigma_j = \begin{cases} 1 & \text{if } b_j=1 \\ -1 & \text{if } b_j=0 \end{cases}$$

Example 1:

Letting t=4 and $2^t + 1 = 17$,

1 0 0 0 0	represents zero
0 1 0 1 0	$= 2^3 - 2^2 + 2 - 1 = 5$
0 0 0 1 1	$= -2^3 - 2^2 + 2 + 1 = -9 = 8 \bmod 17$
and 1 0 1 0 1	is an illegal combination.

Ordinarily, the representation proposed would yield only odd numbers. However, the use of modulo $2^t + 1$ arithmetic means that both even and odd numbers will be represented. To see this, note that

$$B = (2b_{t-1} - 1)\, 2^{t-1} + (2b_{t-2} - 1)\, 2^{t-2} + \ldots + (2b_0 - 1)$$

$$= b_{t-1}\, 2^t + b_{t-2}\, 2^{t-1} + \ldots + 2b_0 - (2^t - 1)$$

$$= (b_{t-2} 2^{t-1} + \ldots + 2b_0 - b_{t-1} + 2) \mod (2^t + 1) \qquad (4)$$

The term in brackets takes on all values from +1 to 2^t and the special case of zero was handled by $b_t = 1$, so all numbers are represented.

Consider arithmetic operations using this number representation. First of all, multiplication by a power of 2 is trivial. If the number is zero (i.e., $b_t = 1$), you do nothing. If the number is non-zero, the low order t bits are circularly shifted to the left a number of places equal to the power of 2, and a bit is replaced by its complement as it enters the LSB position. This is a consequence of the fact that $2^t = -1 \bmod (2^t + 1)$.

Example 2:

Letting t = 4 and $2^t + 1 = 17$, 8 is represented as 0 0 0 1 1. Applying the above rule, 8 x 2 = 16 = 0 0 1 1 1; and 0 0 1 1 1 = 8 - 7 = -1 = 16 mod 17. Further, 8 x 8 = 0 1 1 1 0 = 14 - 1 = 13 = 64 mod 17.

In a hardware implementation the MSB is used as a control bit. If it is one then the number is zero and the rotation is inhibited. This is charac-

teristic of all operations using this new coding system.

Another easy operation is that of forming the negative of a number. Obviously, this is done by complementing the low order t bits except in the case where $b_t = 1$. Again, the MSB is a control bit that would inhibit the operation if it is one. Since we how know how to form the negative of a number and multiply by 2, the only operation left to consider is addition.

If either or both of the operands for addition are zero (i.e., $b_t=1$), then there is no addition to take place; so these special cases can be sensed and the addition inhibited. Now consider the addition of two numbers A and B where $A \neq 0$ and $B \neq 0$. Let

$$A = a_t a_{t-1} \ldots a_0 \quad \text{with } a_t=0,$$
$$\text{and } B = b_t b_{t-1} \ldots b_0 \quad \text{with } b_t=0.$$

Interpret the t LSB's of A and B as the binary representation of $\hat{A}$ and $\hat{B}$, and form the sum of $\hat{A}$ and $\hat{B}$ using unsigned binary addition to obtain $\hat{C}$. That is,

$$\begin{array}{l} \hat{A} = a_{t-1}2^{t-1} + a_{t-2}2^{t-2} + \ldots + a_0 \\ +\hat{B} = b_{t-1}2^{t-1} + b_{t-2}2^{t-2} + \ldots + b_0 \\ \hline \hat{C} = c_t 2^t + c_{t-1}2^{t-1} + \ldots + c_0 \end{array} \qquad (5)$$

It is possible to deduce from $\hat{C}$ the desired sum $C = (A + B) \bmod (2^t+1)$. Since $A = 2\hat{A} + 2 \bmod (2^t+1)$ and $B = 2\hat{B} + 2 \bmod (2^t+1)$, $C = 2\hat{A}+2\hat{B}+4 \bmod (2^t+1)$. If C can be expressed as $C = 2\bar{C} + 2 \bmod (2^t+1)$ with $\bar{C}$ a t bit number, then the t bits of $\bar{C}$ are the t LSB's of C. There are two cases, depending on the value of c_t. If $c_t = 1$, then $\hat{C} = + 2^t + C' = C' - 1 \bmod (2^t + 1)$.

Thus, $C = (2\hat{A} + 2\hat{B} + 4) \bmod (2^t + 1) = (2\hat{C} + 4) \bmod (2^t+1) = (2C'+2) \bmod (2^t+1)$, and $\bar{C} = C'$.

If $c_t = 0$ then $C = 2C' + 4 \bmod (2^t + 1)$ and the answer is $\bar{C} = C'+1$. However, this results in an extra level of add as in the case of 1's complement arithmetic. In 1's complement the output carry is added to the LSB. In this new mod 2^t+1 arithmetic, one takes the output carry, complements it and adds it to the LSB. Thus, the intital claim that this new arithmetic is only as complex as 1's complement is justified. There is a small amount of additional complexity due to the control bit, but this acts only as an inhibit signal.

Example 3:

Let $t = 4$ and $2^t + 1 = 17$.

```
0 1 0 1 0     =     5
0 0 0 1 1     =     8
-----------
  1 1 0 1     =     0 1 1 1 0    =  14 - 1 = 13

0 1 0 1 0     =     5
1 0 0 0 0     =     0        MSB = 1 inhibits the addition
-----------
0 1 0 1 0     =     5

0 1 0 1 0     =     5
0 0 1 0 1     =     5 - 10 = -5 = 12 mod 17
-----------
  1 1 1 1
      + 1
-----------
1 0 0 0 0     =     0 mod 17
```

In this last example note that the second add automatically produced the control bit indicator for the special case of zero. Now let's examine how one converts from a binary coded representation of numbers to this new representation.

3.2 Code Conversion

The code conversion between a binary representation and this new code falls into two cases. Let B be a number which is represented in both codes. Let $b_t b_{t-1} \cdots b_0$ be the binary representation of B and $a_t a_{t-1} \cdots a_0$ the new representation. Also, let $\hat{B}$ be the number represented by interpreting $a_{t-1} a_{t-2} \cdots a_0$ as a binary code. The conversion rules are as follows:

1. If B = 0 then a_t=1, $\hat{B}$ = 0, and b_k = 0 for k = 0, 1, ... t. This is a special case and is done separately.

2. If B ≠ 0 then a_t = 0 and B = $(2\hat{B} + 2) \bmod (2^t+1)$. Conversion from the new code to binary is implemented by forming $2\hat{B} + 2$ and comparing this sum to 2^t. If the sum is larger than 2^t then $2^t + 1$ is subtracted to give the proper binary representation of B.

If the binary representation is given, the sum B + 2^t-1 is formed. If the result is odd, 2^t+1 is subtracted; and finally, this result is right shifted one place. The resulting t bits are the t LSB's $a_{t-1} \cdots a_0$.

Example 4:

Let t = 4 and $2^t + 1 = 17$.

1. B = 1 0 0 0 0 = 16

$$\hat{B} = \left(\frac{16 + 15 - 17}{2}\right) = 7$$

The new representation is B_s = 0 0 1 1 1.

2. $B_s = 0\ 0\ 1\ 0\ 0$

$\bar{B} = 4$

$B = 2 \cdot 4 + 2 = 10 = 0\ 1\ 0\ 1\ 0.$

3. $B = 0\ 0\ 0\ 0\ 0 \longleftrightarrow B_s = 1\ 0\ 0\ 0\ 0.$

In section 5 this new coding scheme will be applied to the logic design of the butterfly of the FNT algorithm. First, the overall structure of the hardware system will be described in Section 4.

IV. System Description

The FNT prototype hardware is a realization of a 64-point transform in the finite field of integers modulo $2^{16} + 1$. The fast FNT algorithm implemented is a radix-2 constant geometry decimation in frequency (DIF) decomposition of the FNT. Figure 1 shows a flow diagram of this algorithm for a 16-point transform (Ref. 7). The constant geometry structure was chosen because it simplifies the memory addressing and the rotation control.

Figure 2 is a block diagram of the complete system showing the four major subsystems. The computational element (CE) is a radix-2 DIF butterfly for the fast FNT algorithm; the memory element contains 128 seventeen-bit words for use as intermediate storage during the transform; the control element is a hardwired implementation of the fast FNT algorithm; and the I/O section provides the interface with the Fast Digital Processor (FDP).

The goal in building this hardware was to construct a CE that would operate at a data rate of 40 MHz. In order to achieve this speed, ECL 10K circuits were used. The basic gate in this logic family has a propagation delay of 2 nanoseconds, and thus these circuits are well suited for very high speed systems. Even with such high speed logic circuits, two levels of reclock and fast carry addition were used in the CE to realize a working system that runs reliably at 38 MHz. In the remainder of this section we will describe the major subsystems of the FNT hardware.

4.1 The Computational Element

The basic computational element of the FNT consists of an adder, a subtractor, and a rotator for multiplication by powers of $\sqrt{2}$. Since

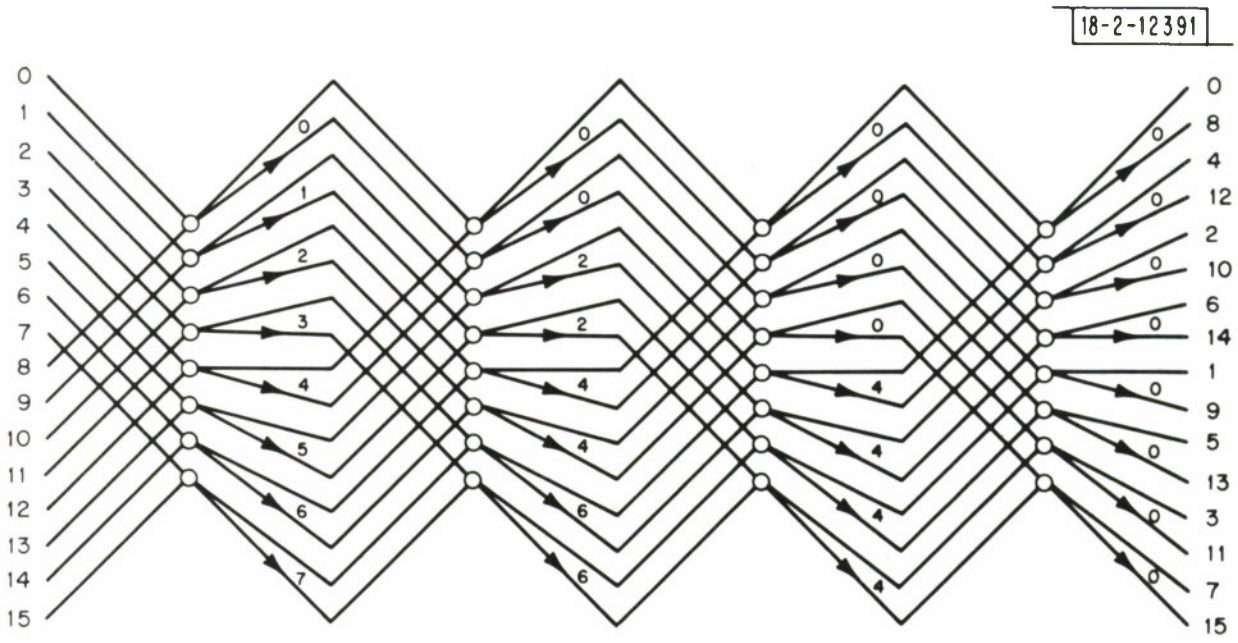


Fig. 1. Radix-2, 16-point, constant geometry FFT (decimation in frequency).

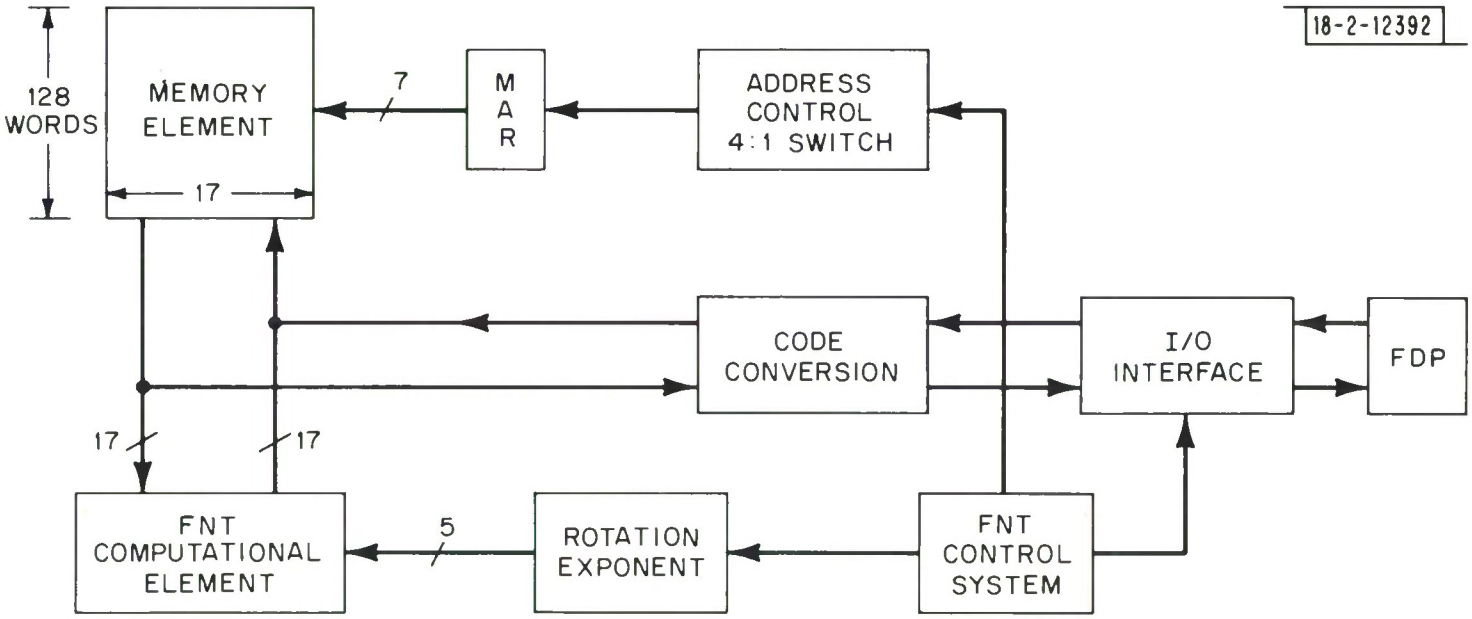


Fig. 2. Block diagram of the 64-point FNT hardware system.

$(\sqrt{2})^k = (\sqrt{2})^{2k_1+k_0} = 2^{k_1}(\sqrt{2})^{k_0}$, where $k_0 = 0$ or 1, the rotator can be divided into two operations: a $\sqrt{2}$ mutiplier (implemented as a subtractor) with a possible inhibit if $k_0 = 0$ and a 2^{k_1} multiplier implemented via a 16:1 multiplexer (see Figure 3). The butterfly must be reclocked twice in order to achieve the 25 nsec clock epoch. Addition (or subtraction) of two 16-bit words modulo $2^{16} + 1$ using carry look ahead addition is just fast enough to fit into the 25-nsec clock epoch. Thus, it is natural to reclock after the first stage of add and subtract and after the $\sqrt{2}$ multiplication. The multiplication by powers of 2 must be implemented using a 16:1 multiplexer arrangement in order to fit into the 25-nsec clock epoch. A shift register implementation would not be economical at such high rates.

4.2 The Memory Element

The memory element contains 128 17-bit words. It was constructed using 17 F10405 128 x 1 ECL RAM's. The access time for these RAM's is less than 15-nsec, so they are well within the speed requirements of the system. For a 64-point transform one only needs 64 words of memory if the transform is done in place. However, in the choice of IC's for the memory element it was economical to select a 128-word chip, and so the constant geometry structure of Figure 1 was employed. In this form of the fast FNT algorithm, the transform is not done in place. Thus, the memory required is twice that of an in place algorithm. However, the memory addressing is simplified because it does not depend on which stage of the transform you are doing. Since there is only one word per memory cell, two memory accesses are required to read the two operands needed for the butterfly. Similarly, two memory write cycles

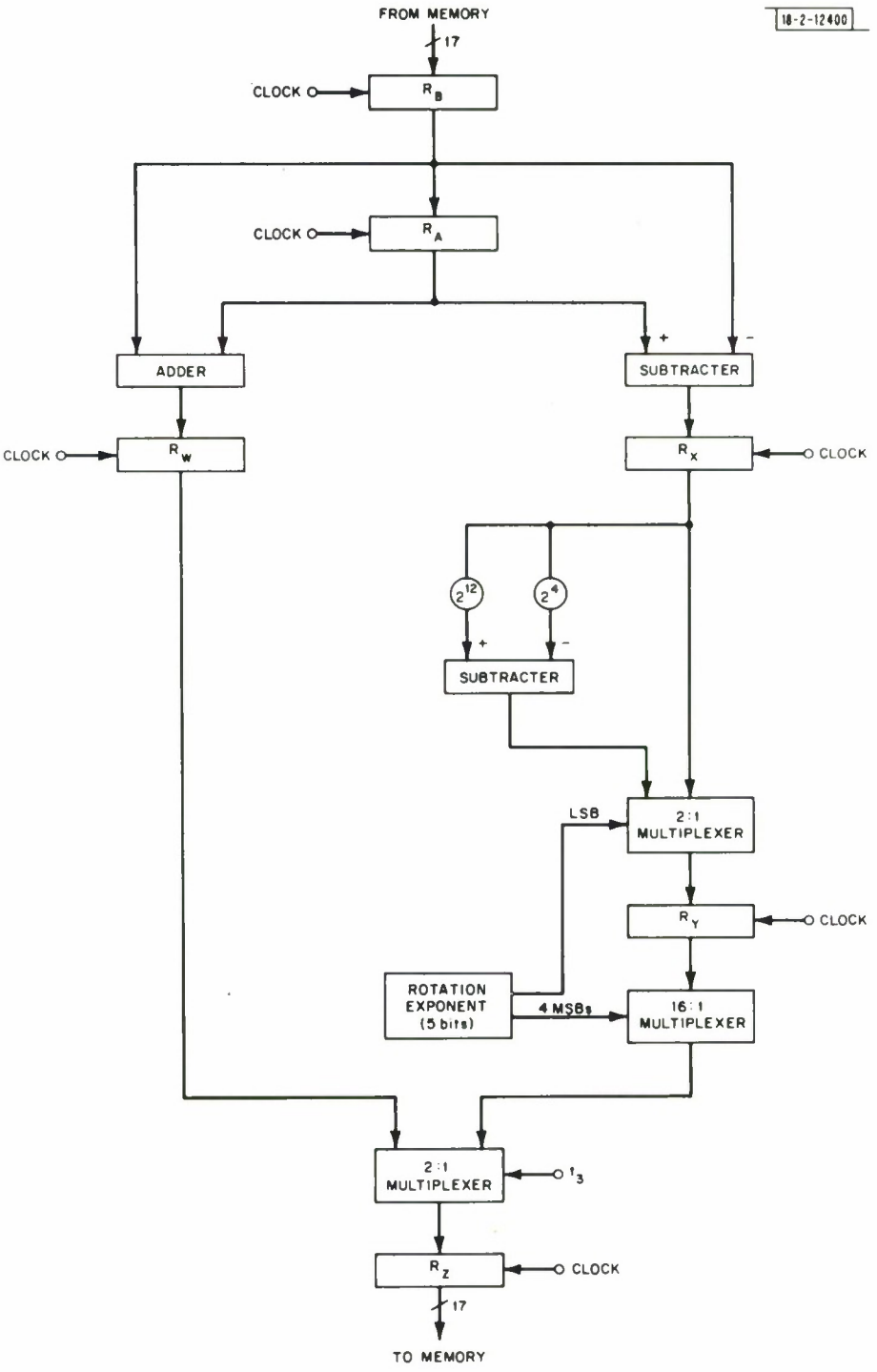


Fig. 3. Block diagram of the computational element of the FNT system.

are required to store the results of a butterfly, and the two data paths are multiplexed into the memory. Thus, a total of four memory accesses are needed in the course of one butterfly. In an actual pipeline structure there would be four distinct interstage delay memories serving one butterfly. Two memories would provide the input operands and two more would acquire the output operands on each clock pulse.

4.3 The Control Element

The control element is the hardware implementation of the FNT algorithm. It controls the reading and writing of memory during the transform and determines the power of $\sqrt{2}$ for rotation. The system operation is broken down into 8 states. Six of these states correspond to the six stages of the transform; the other two states are for synchronization and I/O. When the device is doing a transform the basic timing diagram of the elementary computation is shown in Figure 4. There are six time states, each 25 nsec long. During t_0 and t_1 the two operands A and B needed for the transform are read from memory (see Figure 3). During t_2, A + B and A - B are formed. During t_3, A - B is multiplied by 1 or $\sqrt{2}$ and C = A + B is set up to be written back to memory. During t_4, further rotation of A - B by a power of 2 is done, $D = (A-B) \times 2^k$ is set up for its return to memory, and C is written into memory. Finally, during t_5, D is written into memory. Within each stage a counter keeps track of the fact that 32 butterflies must be executed during each stage and 64 operands must be accessed. The rotation value r is a function of this counter and the stage counter.

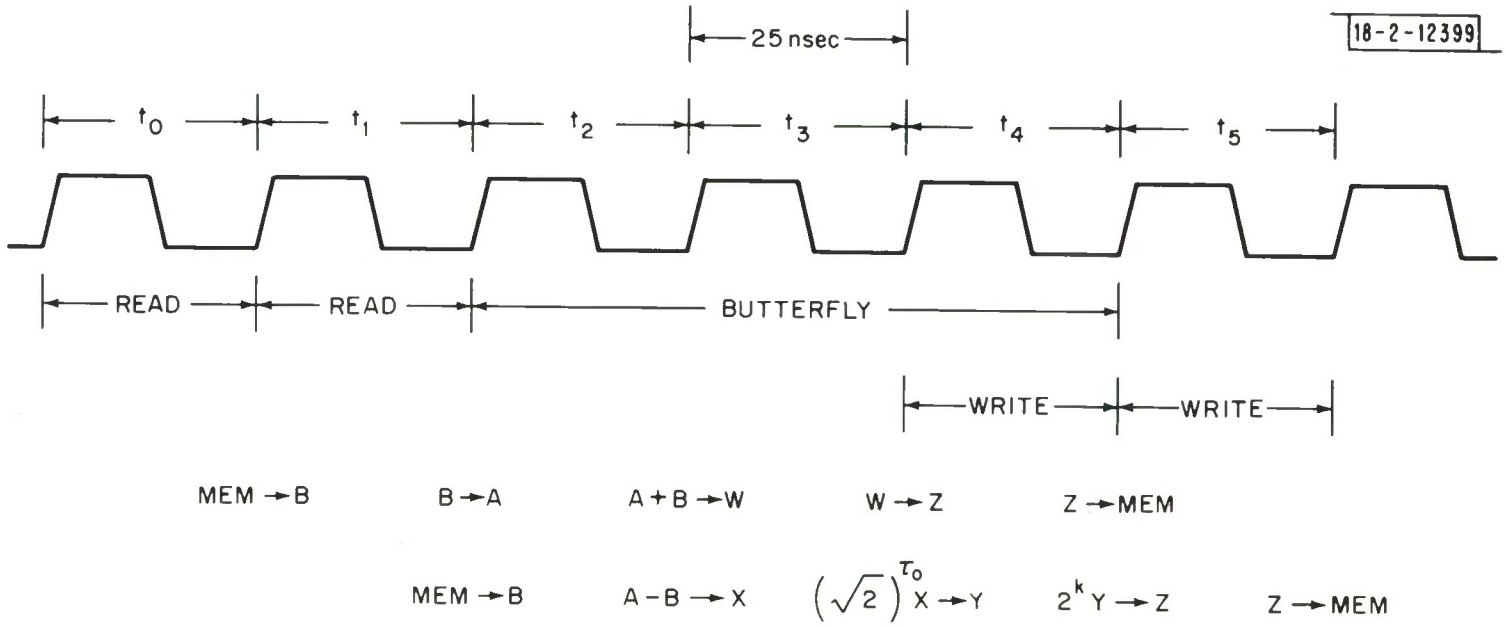


Fig. 4. Timing diagram for the butterfly of the FNT system.

4.4 The I/O Element

The I/O section handles all handshaking with the FDP in order to transfer data back and forth. Communication is always two-way. Thus, whenever the FDP sends a word to the FNT, the FNT is also sending a word to the FDP. The memory element is used as a buffer memory for I/O transfers. Data is transferred in blocks of 64 words. While the 64 data points to be transformed are being sent to the FNT, the 64 output points from the previous transform are being received from the FNT. Data from the FDP is loaded in linear sequence, but it is read out in bit reversed order when being sent to the FDP. Since the result of the transform is in bit reversed order, this bit reversed read out of memory will undo the bit reversal of the FNT. Thus, the data in the FDP is always in normal order.

The code conversion from 2's complement to the code described in section 3 above is also performed in the I/O section before the memory is loaded. A similar conversion back to 2's complement is also done in the I/O section.

V. Logic Design

In this section the logic design of the major elements of the FNT system will be described. The description will concentrate on the arithmetic section which implements the butterfly of the fast FNT algorithm. A basic objective of the logic design was to construct a butterfly which would operate at a data rate of 40 MHz. For this reason ECL 10K logic was employed in the entire system and the butterfly was pipelined with two levels of reclock.

The other subsystems will also be reviewed,but the emphasis will be on their relation to the fast FNT algorithm.

5.1 The Computational Element

Figure 3 shows a functional diagram of the butterfly which consists of an adder, a subtractor, a rotator, input buffer registers R_A and R_B, reclock registers R_W, R_X, and R_Y and an output register R_Z. Register transfers are made at each clock pulse, so that data are always flowing through the CE as would be the case in a pipelined fast FNT. As the timing diagram in Figure 4 shows, the output of the butterfly is only written into memory from R_Z at t_4 and t_5. During the other clock epochs the contents of R_Z may be changing, but this does not affect the algorithm.

5.1.1 Adder (Subtractor) Logic:

In Section 3 a nonstandard coding scheme for data manipulation in the FNT was derived. Recall that the rule for addition of two non-zero numbers $A = \left[a_{16}a_{15} \cdots a_0\right]$ and $B = \left[b_{16}\ b_{15} \cdots b_0\right]$ is:

Step 1: Add the 16 LSB's of A and B with the carry in equal to zero.

Step 2. Complement the carry out from step 1 and add it to the sum from step 1.

If either A or B is zero (i.e., b_{16} or a_{16} = 1) then the carry must be inhibited. Finally if both A and B are zero the MSB of the sum is set to one. Figure 5 shows a realization of the addition process. The structure of Figure 5 is inefficient in two respects. First of all, two 16-bit adders are required, although the second one is simple because one input is zero. Secondly, the addition is very slow because the carry must propagate through both 16-bit adders. The use of carry look ahead logic as in Figure 6 will improve both situations. Now let's look at the details of the implementation using ECL building blocks.

The 16-bit adder can be realized using 4 MC10181 arithmetic logic units (ALU's). Figure 7 shows a block diagram of the 4-bit ALU. In addition to producing the sum outputs, the ALU's also produce carry propagate and carry generate information for use in a carry lookahead block. Thus, the CLA block of Figure 6 is physically spread between the ALU's and a carry lookahead logic unit (MC10179). The addition process can be speeded up further if the carries into each ALU are formed in parallel from the output of the CLA logic. Then the add time will be reduced by 3 x (propagation delay time from C_n to C_{n+4}). For the MC10181 this is approximately 10 nsec which is significant for realizing a 25-nsec clock epoch. Figure 8 shows the final realization of the 25-nsec adder. The logic expressions for the carries were derived from the fact that $C_{n+4} = P_n C_n + G_n$ for each ALU.

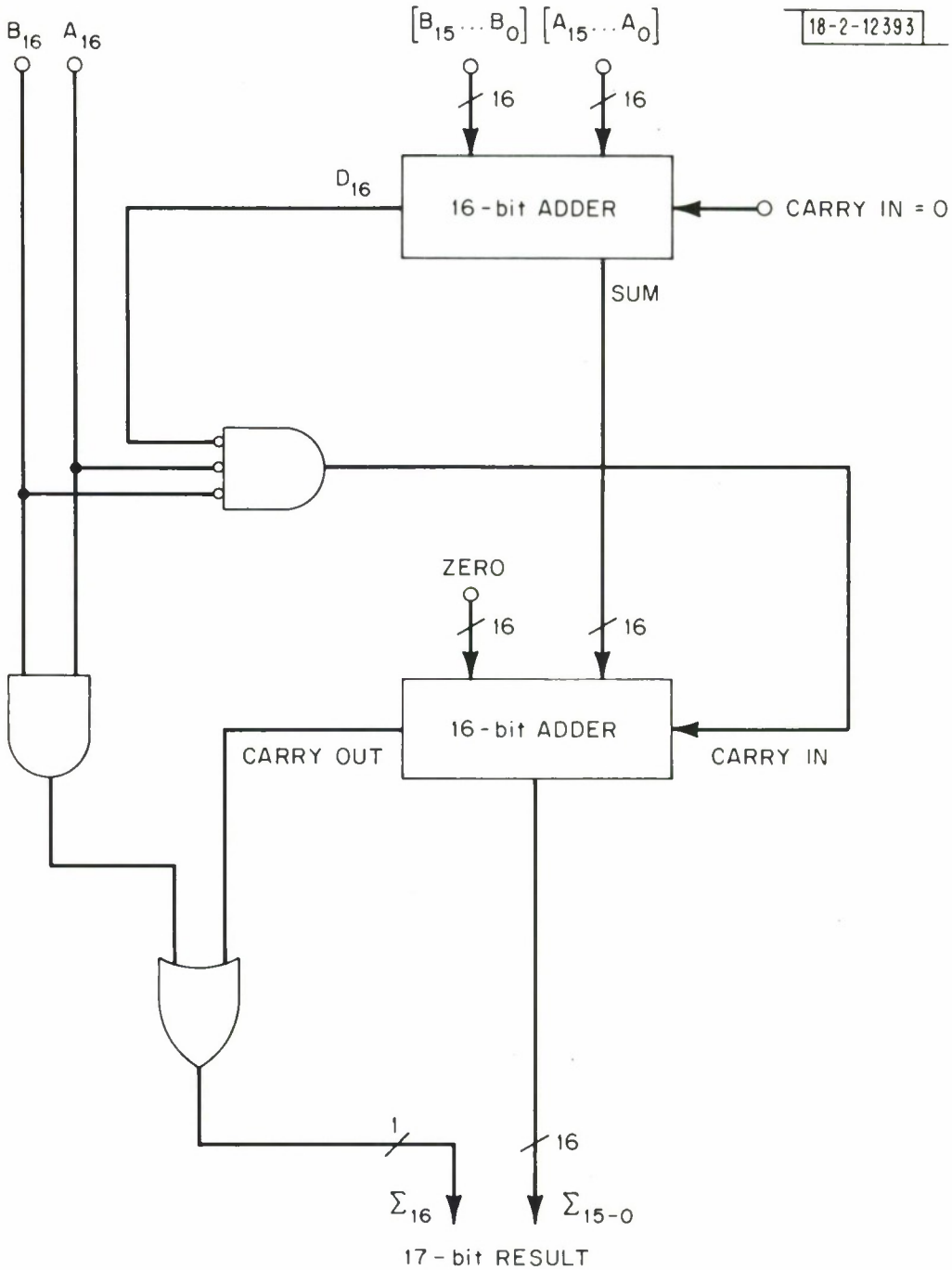


Fig. 5. Fermat number adder (modulo $2^{16}+1$) using two 16-bit adders.

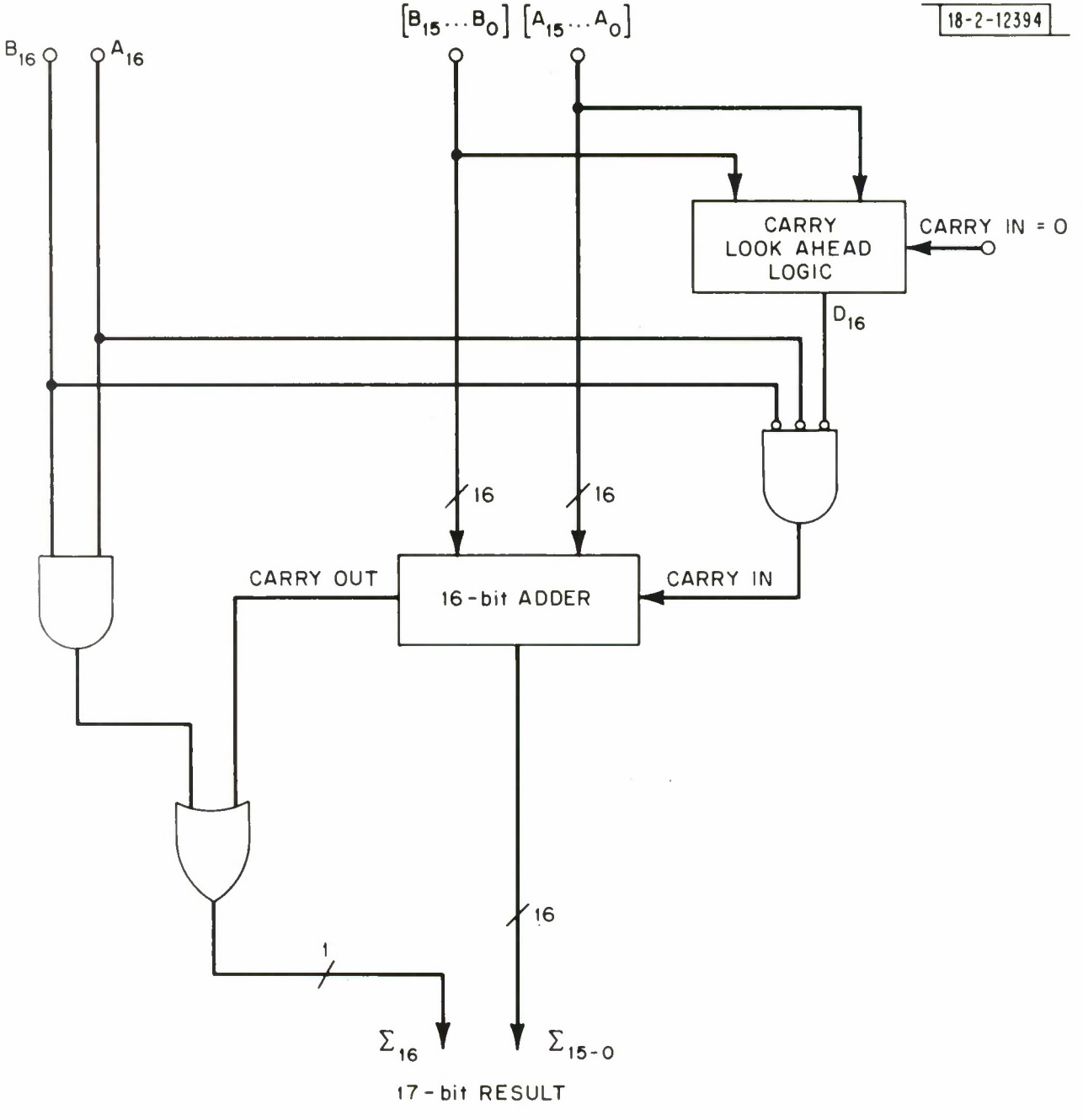
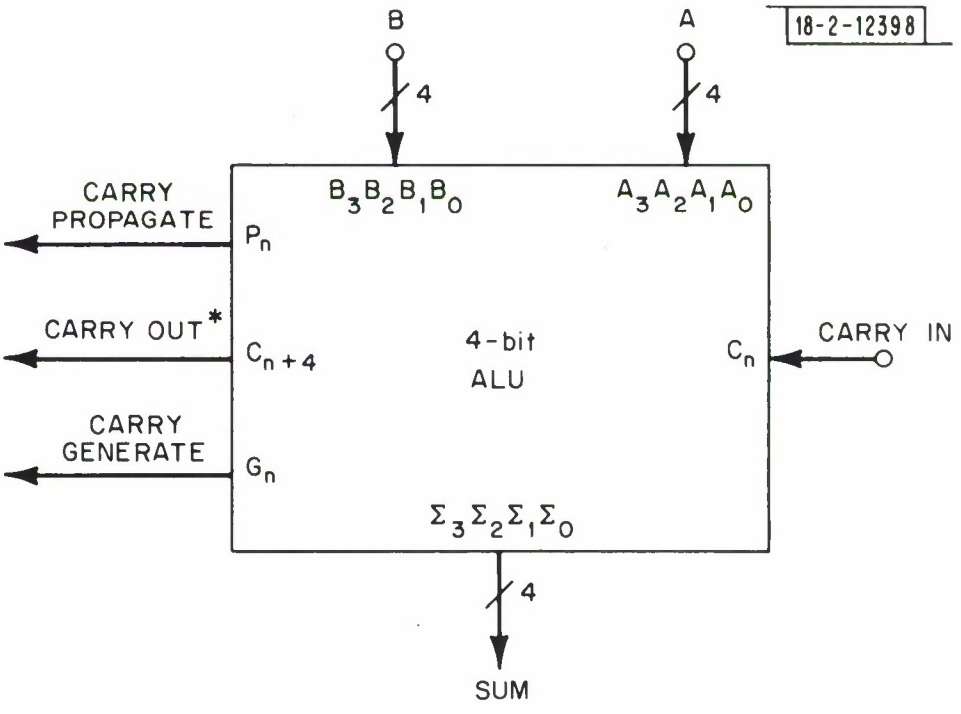


Fig. 6. Fermat number adder implemented with carry look ahead addition.



* CARRY EQUATION: $C_{n+4} = P_n C_n + G_n$

Fig. 7. Functional block diagram of the MC10181 ALU in the addition mode.

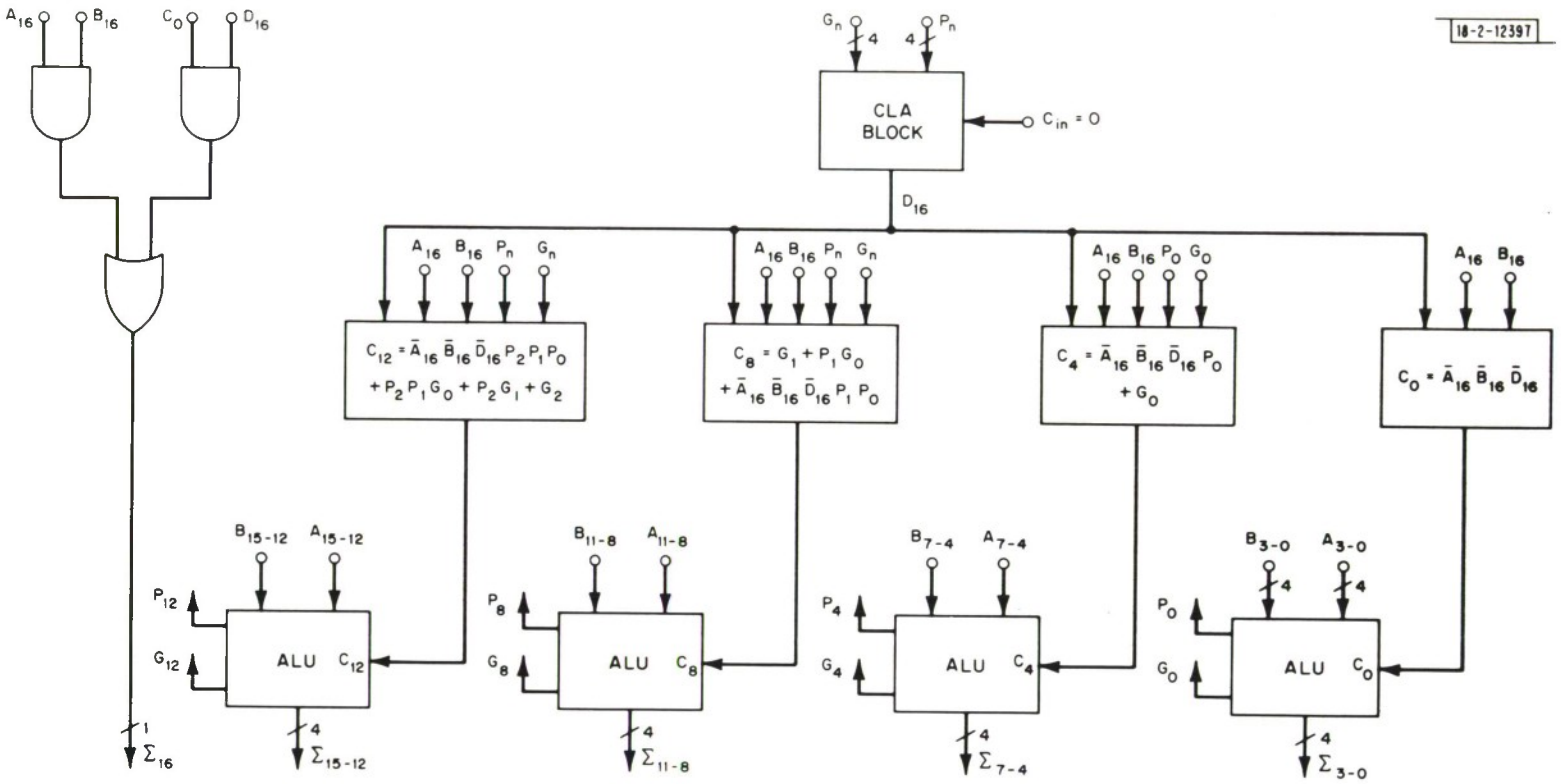


Fig. 8. Logic diagram of the Fermat number adder using fast carry techniques (add time ≃ 21 nsec).

Worst case add time was calculated to be 24 nsec and was measured as 21 nsec. The flip flop setting time and setup time account for the remainder of the 25-nsec epoch.

As noted above the subtractor, A-B can be implemented by complementing B and adding it to A. The use of the MC10181 ALU allows the B input to be complemented internal to the 10181 via mode control. This results in a slight design change in the adder unless the complement is inhibited when B is zero.

The addition A + B completes the calculation on one rail of CE. The result is held in the register R_W and then is moved to R_Z to be written back into memory. On the other rail of the CE, the quantity A-B is stored in the reclock register R_X for subsequent rotation by a power of $\sqrt{2}$.

5.1.2 Rotation by $\sqrt{2}^{\,k}$

The rotation by $\sqrt{2}^{\,k}$ is split into two stages, each requiring a 25-nsec epoch. In the first stage, the quantity X = A - B is multiplied by $\sqrt{2}$ if k is odd. The $\sqrt{2}$ multiplier is merely a subtracter. However, since the output is zero whenever the input is zero, some simplification of the subtractor logic results, and the subtraction time is reduced. A 2:1 multiplexer at the output of the subtractor selects whether the input is to be multiplied by $\sqrt{2}$ or by 1, and is controlled by the LSB of k. The result of this calculation is stored in the reclock register R_Y. The second stage of the rotation is a multiplication by a power of 2, namely $\left[\frac{k}{2}\right]$. ($[\ ]$ denotes the greatest integer function.)

This multiplication is implemented as a 16:1 multiplexer controlled by the upper four bits of the binary representation of the power of 2. Actually the 16:1 multiplexer is realized as a cascade of two 4:1 multiplexers with inverters for the end around shifts. The shifting network is followed by a 2:1 multiplexer which selects which butterfly output ($A + B$ or $2^k \cdot Y$) is to be stored in R_Z and then written back into memory. This multiplexer is controlled by t_3 and its output is $(2^k \cdot Y) \cdot \bar{t}_3 + W \cdot t_3$. This completes the description of the CE; the other elements of the FNT will now be described.

5.2 The Control Element and Memory Element

The control logic is the realization of the Fermat Number Transform algorithm. There are three levels of control and each is driven by a binary counter. The highest level of control consists of 8 states formed from the 3-bit S counter. Six of these states (S_0, $S_1 \ldots S_5$) correspond to the six transform stages; S_6 is a synchronization state; and S_7 is the I/O state.

The second level of control is the indexing within a stage of the FNT. A seven-bit counter, called the K counter, is employed. Within a transform stage or in the I/O state the K counter increments 64 times because each data point must be referenced once. When the 64th count is reached, the S counter is incremented. The K counter is used to form the memory address and the power of $\sqrt{2}$ for rotation. The 7 bits are required for addressing all 128 words of memory.

The lowest level of control is the time state counter called the T counter. The time state counter consists of the six states t_0, t_1, ... t_5 and determines the sequence of operations in the butterfly.

The realization of the FNT algorithm requires the formation of memory addresses and rotation exponents from the three control counters. Letting $K = \left[K_6, K_5 \ldots K_0\right]$, the rotation exponent (called twiddle control) is $\tau = \tau_4 \tau_3 \ldots \tau_0$ where

$$\tau = \begin{cases} K_5\ K_4\ K_3\ K_2\ K_1 & \text{when} \quad S = (0\ 0\ 0) = s_0 \\ K_5\ K_4\ K_3\ K_2\ 0 & S = (0\ 0\ 1) = s_1 \\ K_5\ K_4\ K_3\ 0\ 0 & S = (0\ 1\ 0) = s_2 \\ K_5\ K_4\ 0\ 0\ 0 & S = (0\ 1\ 1) = s_3 \\ K_5\ 0\ 0\ 0\ 0 & S = (1\ 0\ 0) = s_4 \\ 0\ 0\ 0\ 0\ 0 & S = (1\ 0\ 1) = s_5 \end{cases}$$

The memory address is formed in one of four ways depending on the counters. When memory is being written, the MAR is equal to K, and the K counter is incremented after the write. When the memory is being read during the transform, there are two possible memory addresses. During t_0, the MAR is $\left[\overline{K}_6\ 0\ K_5 \ldots\ K_1\right] = \rho_0(K)$; during t_1, the MAR is $\left[\overline{K}_6\ 1\ K_5 \ldots\ K_1\right] = \rho_1(K)$. Finally, during I/O the memory is used in bit reversed order and the MAR equals $\left[\overline{K}_6 K_0 \ldots\ K_5\right]$.

In order to complete the specification of the control of the FNT, a register transfer sequence is provided below. This corresponds to the timing diagram in Figure 4.

		Comments
$t_0 \cdot CL:$	$MAR = \rho_o(k)$	Read operand A from memory
	$MDR \rightarrow R_B$	
$t_1 \cdot CL:$	$MAR = \rho_1(k)$	Read operand B from memory
	$R_B \rightarrow R_A$	
	$MDR \rightarrow R_B$	
$t_2 \cdot CL:$	$SUM\ (A, B) \rightarrow R_W$	A + B and A - B
	$DIFF\ (A, B) \rightarrow R_X$	
$t_3 \cdot CL:$	$R_W \rightarrow R_Z$	$\sqrt{2}$ Multiplication
	$\tau_o \cdot MUL\ (\sqrt{2}, X) + \bar{\tau}_o \cdot R_X \rightarrow R_Y$	
$t_4 \cdot CL:$	$MAR = K$	Write Memory
	$K + 1 \rightarrow K$	
	$R_Z \rightarrow Mem$	
	$ROT\ (\tau, Y) \rightarrow R_Z$	Multiply by 2^k
$t_5 \cdot CL:$	$MAR = K$	Write Memory
	$K + 1 \rightarrow K$	
	$R_Z \rightarrow Mem$	

5.3 The I/O Element

The I/O element was designed to provide asynchronous transfer of data between the FNT system and the FDP. The I/O is enabled only when the S counter is in state s_7. An input request (IR) from the FDP is first synchronized to the FNT clock and then the following I/O sequence takes place:

1. The memory is read with the bit-reversed address, MAR = $\left[\bar{K}_6\ K_0 \ldots K_5\right]$. The output is stored in register R_E. The output of R_E is code converted to 2's complement and is transmitted to the FDP.

2. The input acknowledge line (IA) is raised approximately 50-100 nsec after the acceptance of the IR.

3. Data are written into the memory after being code converted from 2's complement to the new code used internal to the FNT. This memory write occurs approximately 250 nsec after the acceptance of the IR. The K counter is incremented after the write.

4. The IA is cleared after 1.2 μsec and is held down for a minimum of 200 nsec.

The data rate achievable with this interface to the FDP is approximately one word every 1.5 μsec.

VI. FDP Peripheral

In this section the operation of the FNT system as an FDP peripheral device will be described. Two topics will be discussed: the operation of the manual controls of the FNT and the constraints to be observed by FDP programs that use the FNT.

Figure 9 shows the layout of the control panel of the FNT. The function of the switches and lights is as follows:

1. S/R Switch: The Stop/Run switch is a two position switch. When the switch is in the down position the FNT is in the run mode and normal (full speed) operation of the hardware is enabled. When the switch is placed in the up position, the FNT is stopped. The stop condition is indicated by the red light above the S/R switch. In the stop mode, depressing the CYCLE button will step the machine by one clock cycle. This feature was used for debugging the hardware and shouldn't concern the programmer.

2. The MEM.T switch is a two position switch that will allow the memory element of the FNT to be tested. When this switch is in the down position, the FNT is in transform mode. When the switch is in the up position, the device is in the memory test mode and the butterfly is disabled. A red light above the MEM.T switch signifies that the FNT is in memory test mode. In this mode data transferred from the FDP to the FNT are returned to the FDP unchanged except for a bit reversal. This bit reversal results from the I/O addressing modes described in Section V. A diagnostic program uses this mode to check for the proper operation of the memory.

3. The RESET button is used to initialize the FNT machine. When the button is depressed, all counters are initialized and the system is put in the

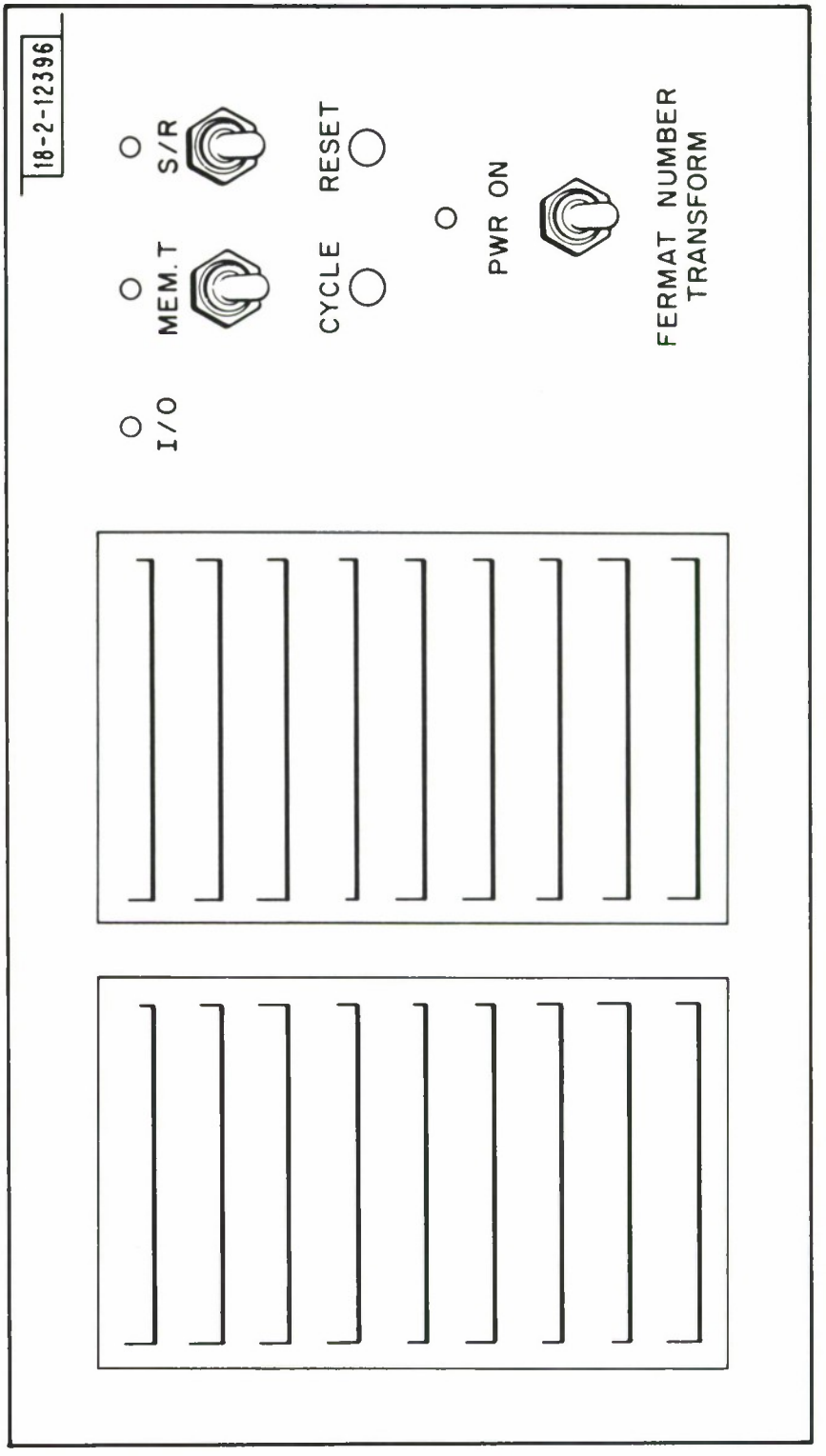


Fig. 9. Front panel of the FNT machine.

synchronization state. For proper initialization the FNT should be stopped when a RESET is done.

4. The CYCLE button, when pushed, will step the machine by one clock pulse if the FNT is stopped. If the machine is in RUN mode, the CYCLE button has no effect.

5. The I/O light signifies that the FNT is in the I/O state where it is waiting for I/O with the FDP.

Initialization of the FNT for use in the transform mode requires the following steps.

(i) Place MEM.T switch in the down position. The red light above the switch should be off.

(ii) Place the S/R switch in the up position. The red "Stop" light should come on.

(iii) Press the RESET button and hold in for 1 or 2 seconds. The I/O light should be off after this operation.

(iv) Place the S/R switch in the "run" position (i.e., down). The "stop" light should go off and the I/O light should come on. Now the device is initialized and ready to accept data from the FDP. We now turn to a discussion of the programming features of the FNT.

The FNT is connected to the FDP via subchannel 6 of I/O channel 1. Since data communication between the two machines is always full duplex, the I/O handshaking is done using the control signals of the FDP input channel. Thus, the execution of an IOC instruction which sets the input request line will cause the FNT to take as input the contents of the FDP E_o register and to send an output word which will be strobed into the E_i register of the FDP. The timing of the I/O

transfers allows E_o to be loaded in the same instruction as the IOC. Thus, the following instruction is valid.

```
TME DATA 0  0          IOC 3 3 0 0 15
```

In order to do a transform 64 data points must be loaded into the FNT. The FNT machine automatically starts the calculation of the transform when the 64th data point arrives. At the present 38-MHz clock rate, the transform takes about 30 μsec. To obtain the results of the transform 64 more I/O transfers must be done. The full duplex mode of operation allows one to load data for a new transform and obtain the results of the previous transform at the same time. Since the I/O of 64 data points takes 2 or 3 times as long as the transform itself, this is an important factor for obtaining maximum performance from the machine.

In order to do convolution using the FNT hardware, it is necessary to do the reference spectrum multiplication (mod $2^{16} + 1$) in the FDP. A possible program to do this is given below. The program will multiply the two 64-point arrays DATA (MA) and DATA (MB) modulo ($2^{16} + 1$) and store the result in DATA (MB). Since the FNT hardware will only compute a forward transform, a reordering of the data in the FDP is necessary to obtain an inverse transform. The 64-point array x(k) to be inverse transformed must be flipped according to the formula

$$\tilde{x}(k) = x\ (\ <64 - k> \bmod 64)$$

Then the forward FNT of $\tilde{x}(k)$ is the inverse FNT of x(k).

```
(PROGRAM TO DO REFERENCE MULTIPLY FOR 16 BIT FNT
(MULTIPLIES DATA(MA) BY DATA(MB)
(PUTS RESULT IN DATA(MB)
(DOES FOUR MULTIPLIES PER PASS
(FERMAT(MA) = 200001
(FERMAT(MB) = 177777

MULUP                                   YIX FERMAT 10
                                        YIX 63. 7
        AI1BI2 DATA 7 7                 YPX -1 7
MULUP0  AI3BI4 DATA 7 7                 YPX -1 7
        /MIF//MIF                       /TIQ//TIQ
        BI24 0 10                       /MUL//MUL
        AI3BI4 DATA 7 7                 /TIQ//TIQ
        /TPR//IPR                       YPX -1 7
        AI1BI2 DATA 7 7                 YPX 3 7
        MIB/TDR/MIB/TDR                 TIQ/TRI/TIQ/TRI
        MUL/TRI/MUL/TRI                 /IAQ//IAQ
        AI13BI24 0 10 10                /RMI//RMI
        MIB/MIF/MIB/MIF                 JNR FX2 4
        TPR//TPR/                       TIQ//TIQ/
        R2B DATA 7                      JNR FX4 1
        TRI//TRI/                       TDR//TDR/
        IAQ//IAQ/                       TRI//TRI/
        RMI//RMI/                       YPX -1 7
        R4B DATA 7                      JNR FX4 2
        /MRF//MRF                       YPX -1 7
                                        JNR FX2 4
        R4B DATA 7                      YPX -1 7
        R2B DATA 7                      JPX MULUP0 7
        AI1BI2 DATA 7 7                 YPX -1 7
        SOJ 0 2                         XJP -1 RTRN

FX2                                     XJP 0 1
                                        /IPR//

FX4                                     XJP 0 1
                                        ///IPR


        END
```

VII. Diagnostic Programs

In order to debug and maintain the FNT hardware, three diagnostic programs are available:

1. Memory Test
2. Rotator Test
3. Adder/Subtractor Test

Each diagnostic program tests a separate functional element of the hardware, and the tests form a hierarchy in that each diagnostic test relies on the proper operation of the functional element tested by the previous diagnostics.

The memory test is used with the hardware in the memory test mode. The purpose of the test is to check the memory element of the FNT and the data communication link between the FDP and the FNT. Data words are generated in the FDP and sent to the FNT in blocks of 64. Then as a block of 64 words is being sent, the words being received from the FNT are compared to the previous block sent and checked for errors. If an error occurs, the FDP halts with information concerning the error in the AE lights. Successful completion of the memory test allows one to test the FNT in its transform mode with the remaining two diagnostic tests.

Both of the transform tests use known transform pairs to test different parts of the FNT butterfly. In the rotator test the rotation element of the CE is subject to test. The following transform pair is used.

$$\text{FNT}\left\{e^{(r)}\right\} = x^{(r)} = \left[1 \quad (\sqrt{2})^{r} \quad \ldots \quad (\sqrt{2})^{r(n-1)}\right]^{T}$$

where $$e^{(r)} = \left[0 \ \ldots \underset{\substack{r^{th} \\ \uparrow \\ \text{position}}}{1} \ \ldots \ 0\right]^{T}$$

Since the transform of $e^{(r)}$ involves no additions or subtractions (except with zero), this test serves to check out the rotation element of the butterfly. As before, the program halts if an error is detected and displays the erroneous data in the AE lights.

In order to check the adder/subtractor pair in the butterfly, another known transform pair is used.

$$\text{FNT}\left\{\left[a\ 0\ \ldots\ 0\ \underset{\substack{\uparrow\\ 32\text{nd position}}}{b}\ 0\ \ldots\ 0\right]^{T}\right\} = \left[a+b,\ a-b,\ a+b,\ldots,a-b\right]^{T}$$

The first two elements of the transform are examined to check the addition and subtraction. The numbers a and b are varied to check all possible cases. The successful completion of all three diagnostic tests should guarantee that the FNT hardware is operating properly.

VIII. Hardware Comparison of the FNT vs the FFT

Since one purpose in building a hardware prototype of the FNT was to gain a working knowledge of the amount of hardware needed to implement an FNT convolver, it is appropriate to compare this method with the standard FFT implementation of convolution. It is impossible to make a comparison that will apply in all cases, or even a majority of cases. Therefore, a specific application has been chosen for comparison; namely, digital filter implementation for radar signal processing. The problem areas to be described below should be representative of the general problems associated with FNT convolution implementation.

The signal bandwidths encountered in radar signal processing (10-30-MHz) require a pipeline architecture for either the FNT or the FFT [8]. Furthermore, the length of the convolution to be implemented is assumed to be large (e.g., 512 or greater). Two cases will be considered: a length 1024 linear convolution of real data and a length 1024 convolution of complex data. Four measures of hardware complexity are the basis of comparison: the number of butterflies per output point, the number of reference spectrum multiplies per output point, the total amount of interstage delay line memory in the forward and inverse transforms, and the total amount of reference spectrum memory. The FFT implementation will be considered first.

For either real or complex data, it is assumed that the FFT implementation employs an 11-stage radix-2 pipeline FFT in both the forward and inverse transforms. (Note: for real data it is possible to do a length N transform with two length N/2 transforms and some overhead to combine the two shorter transforms [9]. However, the overhead amounts to an additional butterfly so

there is little, if anything to gain using this fact in a pipeline FFT). Much work has been done on the hardware complexity of pipeline FFT's and the four measures of complexity we are considering here are detailed in ref. 10 . For the case of a length 2048 pipeline FFT convolver the number of butterflies per output point is $2 \log_2 N = 22$, assuming 50% convolutional overlap. Likewise, two reference spectrum multiplies must be done per output point. The amount of interstage delay line memory can be calculated from the formula

$$IDM = \frac{N}{2} (r + 1) \tag{6}$$

where r is the radix of the transform. Thus, for two radix -2 pipelines, the total is 3N = 6144 = 6K words of memory. Finally, the reference memory requires 2K words of memory. Now we turn our attention to the FNT.

A pipeline FNT structure is identical to the pipeline FFT except in the butterfly where rotation by $e^{j2\pi k/N}$ (in the FFT case) is replaced by multiplication by $\sqrt{2}^{\,k}$. Thus, many of the results quoted above are applicable to the FNT. Since the FNT naturally processes real input data, the cases of real and complex convolution require different realizations. In both cases, however, a two-dimensional implementation of the convolution is required [4]. The two arrays to be convolved are H and X, where

$$X = \begin{bmatrix} x(0) & x(L) & \cdots & x(N-L) \\ x(1) & & & \\ \vdots & & & \\ x(L-1) & x(2L-1) & \cdots & x(N-1) \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & 0 \end{bmatrix} \quad \begin{matrix} \\ \\ \\ \\ 64 \\ \\ \\ \end{matrix} \qquad \begin{matrix} \\ \uparrow \\ L = 32 \\ \downarrow \\ \\ \\ \end{matrix} \tag{7a}$$

$\leftarrow$ 64 $\rightarrow$

and

$$H = \begin{bmatrix} h(n-L+1) & h(1) & \cdots & h(N-2L+1) \\ \vdots & \vdots & & \vdots \\ h(N-1) & h(L-1) & & \vdots \\ h(0) & h(L) & & \vdots \\ h(1) & . & & \vdots \\ \vdots & . & & \vdots \\ h(L-1) & . & & \vdots \end{bmatrix} \qquad (7b)$$

The length 1024 convolution of real signals can be implemented with a 64 x 64 transform. The input data is the array X of equation 7 and it is advantageous to exploit the fact that half of the X array is zero by doing the row transforms first. Hence, 96 length 64 FNT's must be computed for the entire 2-D transform. The total number of butterflies for the complete convolution is $2 \times 96 \times 32 \log_2 64 = 9 \times 2^{12}$, or 36 butterflies per output point. This is reflected in the structure of Figure 10 where there are 36 butterflies working in parallel -- six in each 64-point FNT. Since 2^{12} reference function multiplies must be done during each convolution, four reference multiplies per output point are required. The interstage delay memory requirement is calculated from the individual 64-point transforms. The first and last 64-point FNT's are computing 32 transforms simultaneously. This is accomplished by making all interstage delay lines 32 times as long as in a standard pipeline and by modifying the control to switch at 1/32nd the speed of a normal pipeline. That is, the rotation exponents and commutator switches are only changed at every 32nd clock pulse. The other four FNT's employ a normal pipeline structure. Applying equation 6, the total interstage delay memory is $3 \cdot 64 \cdot 32 + 2 \cdot 3 \cdot 64 = 6K + 384 \cong 6.4K$

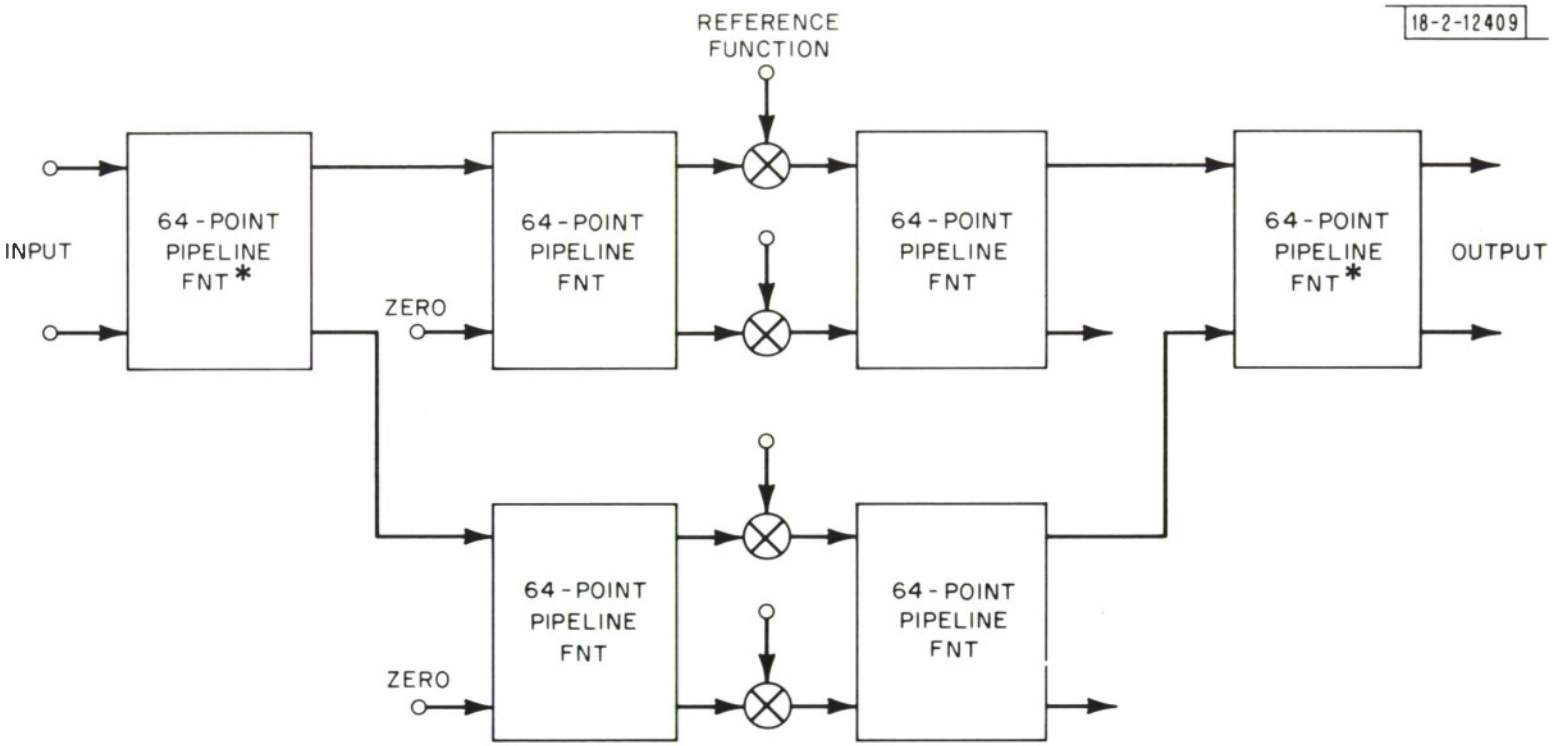


Fig. 10. Pipeline realization of a two-dimensional (64 x 64) FNT convolution.

Finally, it is easy to verify that the amount of reference function memory is 4K.

If the signals to be convolved are complex then one possible implementation is to handle the real and imaginary parts of the data separately. The number of butterflies per output point, the interstage delay memory, and the reference spectrum memory are all doubled. However, the number of real multipliers for the reference function multiply is quadrupled because the multiplication is complex. Table 1 summarizes the three systems versus the four hardware measures. Notice that the FNT always requires more memory and more computational elements than the FFT. Hardware savings are possible because most of the hardware cost of the FFT is concentrated in the butterfly elements (up to 80%) and because the FNT butterfly requires from one-third to one-sixth the hardware of an FFT butterfly. These remarks apply to the FNT when the data to be convolved arereal, but when the data are complex the situation becomes much more difficult because all measures of hardware complexity are increased by a factor of three or four. Therefore, we will concentrate on the case of real convolution in the following discussion.

In order to be more specific, let's divide the hardware cost of any convolver into two parts, the percentage of hardware for the butterfly elements and the percentage of hardware for the rest of the system. Assume the following relations between the FFT and the FNT

$$B_{FNT} = 36/22 \ \lambda \ B_{FFT} \tag{8}$$

$$B_{FFT} = \mu T_{FFT} \tag{9}$$

$$(T_{FNT} - B_{FNT}) = \nu \ (T_{FFT} - B_{FFT}) \tag{10}$$

Table 1. Hardware measures for FFT and FNT implementations of length 1024 convolution.

	FFT Real or Complex Data	FNT Real Data	FNT Complex Data
Butterflies	22	36	72
Reference Multiplies	2	4	16
Interstage Delay Memory	6K complex words *	6.4K real words **	12.8K real words
Reference Spectrum Memory	2K complex words	4K real words	8K real words

* One complex word will contain approximately 25 bits for typical high-precision radar applications.

** One real word contains 33 bits for FNT

where B denotes the butterfly cost and T the total convolver cost. Letting $T_{FFT} = 1$ the total hardware for the FNT is

$$T_{FNT} = \left(\frac{18}{11}\lambda - \nu\right)\mu + \nu \tag{11}$$

and the savings can be expressed as

$$S_{FNT} = 1 - T_{FNT} = (1 - \nu) + \mu\left(\frac{18}{11}\lambda - \nu\right) \tag{12}$$

Thus, the problem is now that of determining realistic values for the three ratios λ, μ and ν. If we assume that the accuracy requirements of the system are high, then the FFT implementation should use a hybrid floating point scheme as described in reference 6. This study described a complex data format using 27 bits -- 11 bits each for the real and imaginary parts of the mantissa and 5 bits for a common exponent. In contrast the FNT would employ modulo $2^{32} + 1$ arithmetic, implying a 33-bit data word. From this information we would like to argue that a reasonable value for ν is two. Three components must be considered. First, the numbers of reference multiplies being compared are two 11 x 11 complex multipliers and four 32 x 32 real multipliers. Since the hardware complexity of an array multiplier (which would be required at radar speeds) is proportional to n^2, the four real multipliers amount to about $\frac{4 \cdot 3^2}{2 \cdot 4} = 4\frac{1}{2}$ times the hardware of the two complex multipliers. Secondly, the interstage delay memory is 6K words of 27 bits each for the FFT versus 6.4K words of 33 bits each for the FNT, or a factor of 1.3 in favor of the FFT. Similarly, the reference function memory differs by a factor of 2.44 in favor of the FFT. Depending on the detailed logic realization, the value of ν will vary, but

$\nu = 2$ conveys the fact that the two-dimensional convolution using the FNT wastes a factor of two in non-butterfly hardware. This is a fundamental limitation of the FNT for long convolutions. At this point we can isolate the impacts of the butterfly hardware on the possible savings for the FNT. Figure 11 shows a plot of hardware savings versus the percentage of butterfly hardware in the FFT (μ). Several curves are shown with λ as a parameter. λ is the ratio of FNT butterfly hardware to FFT butterfly hardware. The range of values for λ are typical for the high speed implementations required in radar signal processing. The value of ν was assumed to be two and if it were larger then the horizontal intercept would be moved to the right to a higher value of μ.

Independent of the exact values of λ, μ and ν, Figure 11 clearly shows that the FNT will provide hardware savings over the FFT only when the FFT hardware is dominated by the CE cost as in the case of a pipeline implementation.* Furthermore, the signals to be convolved must be real, because complex data essentially require two real FNT convolvers. Although short convolutions (e.g., length 64) have not been discussed here, it is worth mentioning that there is a good chance for significant hardware savings over the FFT because a one-dimensional FNT can be used. This means that the non-butterfly hardware of the two systems will be approximately the same (i.e., $\nu \approx 1$) and the savings will begin at a value of μ near zero.

---

*Note: As transform size increases the memory cost of a pipeline increases faster than the CE cost and will become a significant fraction of the total cost for large transforms (e.g., length 16K or 32K).

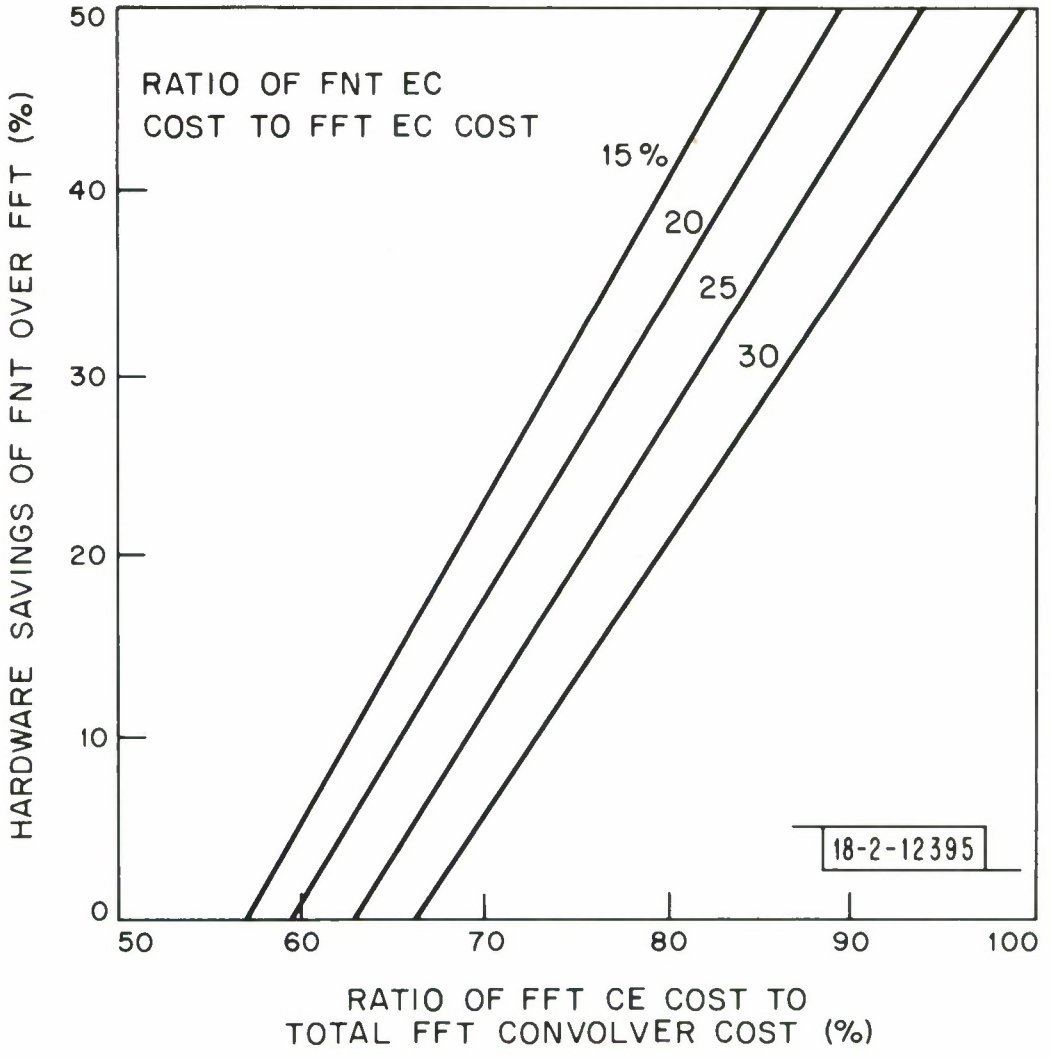


Fig. 11. Region of potential hardware savings of the FNT over the FFT for a length 1024 aperiodic convolution of real signals.

IX. Examples of Filter Implementation

In order to demonstrate the capabilities of the FNT hardware, two filters were implemented and the results are shown below. The first filter chosen was a length 33 FIR lowpass filter with a passband cutoff frequency of 0.10, stopband cutoff frequency of 0.15 and stopband attenuation equal to -33 db. The second filter was a length 33 bandpass filter with -44.5 db attenuation in the stopbands. The cutoff frequencies of the bandpass filter were 0.15 for the lower stopband, 0.27 for the upper stopband and 0.20 to 0.22 for the passband. Figures 12a and 12b show the ideal frequency responses of the two filters obtained using an FFT of the impulse responses. The frequency response is shown from F = 0 to F = 0.5 (assuming the sampling frequency is unity) and the horizontal lines denote steps of 20 db.

The method chosen to show the FNT implementation of convolution was to filter a discrete time linear FM signal with the lowpass and bandpass filters. The output of the filters traces out an approximate frequency response of the filter, because the input linear FM sweeps across the frequency range of interest. Figures 13a and 13b show the results of the convolution when the input signal is represented with 7 bits. For the lowpass filter it is possible to use 8 bits for the filter coefficients without overflowing the computation but the bandpass filter can use only 7 bits. The response of the lowpass filter is good approximation to the ideal response of Figure 12a and much of the roughness can be attributed to the fact that the linear FM only traces out an approximate spectrum. The bandpass filter (Figure 13b) is much worse because specifications of the filter are more stringent and one less bit is available

for filter coefficients. The conclusion one reaches is that the FNT based on $F_4 = 2^{16} + 1$ is marginal for the implementation of most filters.

It is possible to extend the precision of the FNT implementation in several ways. One possibility is a method based on the Chinese Remainder Theorem [5]. In this method, two convolutions are computed modulo two relatively prime numbers and the results are combined term by term using the Chinese Remainder Theorem. In particular, $\{h_k\}$ and $\{x_k\}$ are convolved modulo $(2^{16} + 1)$ to obtain $\{\tilde{Y}_n\}$ and modulo 2^5 to obtain $\{\hat{Y}_n\}$. Assuming the true output $\{Y_n\}$ satisfies $Y_n < 2^5 \cdot (2^{16} + 1)$ then Y_n can be written

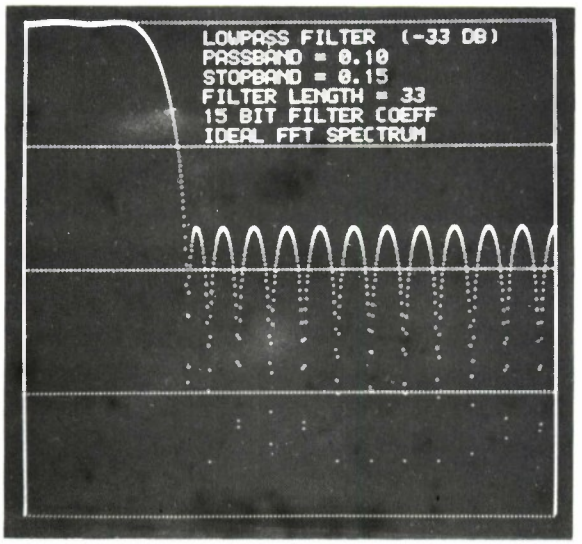
$$Y_n = \tilde{Y}_n + f \cdot (2^{16} + 1)$$

Then assuming $f < 2^5$, we have

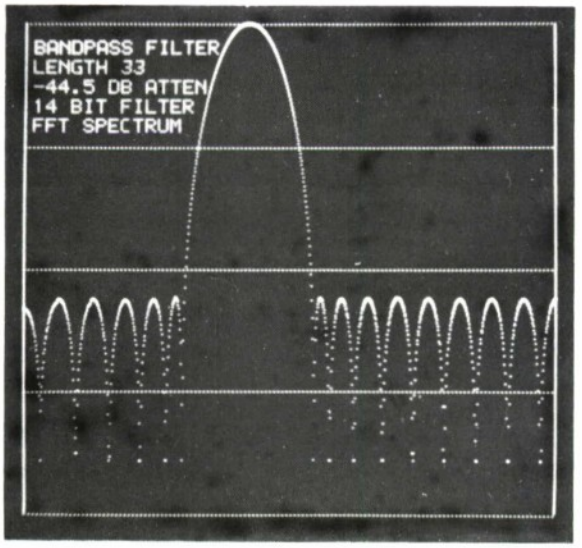
$$f = (\hat{Y}_n - \tilde{Y}_n) \bmod 2^5$$

Note that the convolution of $\{h_k\}$ and $\{x_k\}$ modulo 2^5 can be done with the FNT hardware by convolving $\{h_k \bmod 2^5\}$ with $\{x_k \bmod 2^5\}$ and taking the result modulo 2^5.

Thus, by applying the Chinese Remainder Theorem, 5 extra bits of precision are available for the filter implementation. Figures 14a and 14b show the response to linear FM of the lowpass and bandpass filters with 11-bit filter coefficients. In both cases, the response is very near the ideal of Figure 12 with the slight differences due to the linear FM input signal.

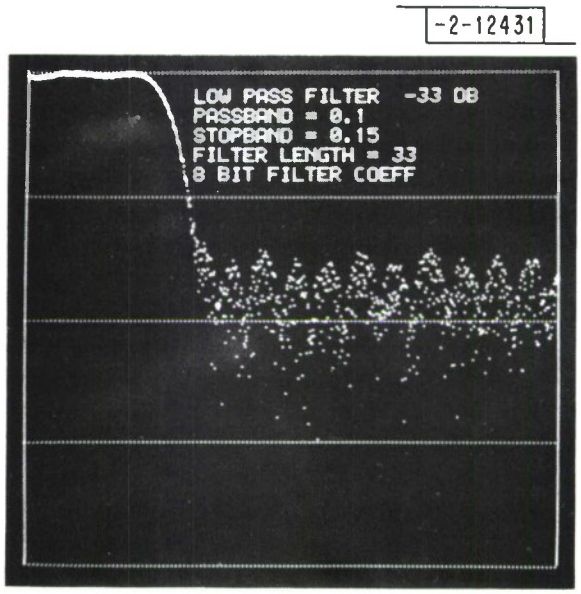


(a)

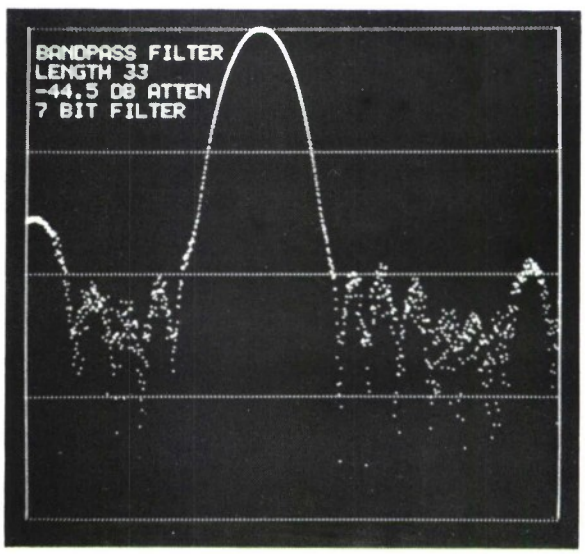


(b)

Fig. 12. Ideal filter responses.
a) lowpass filter
b) bandpass filter

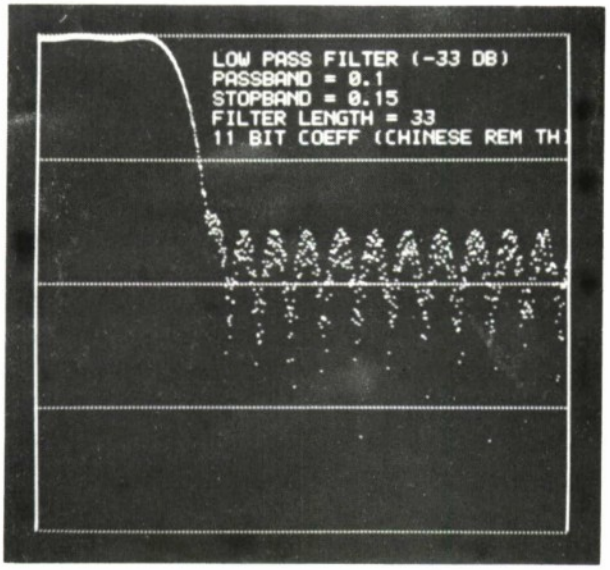


(a)

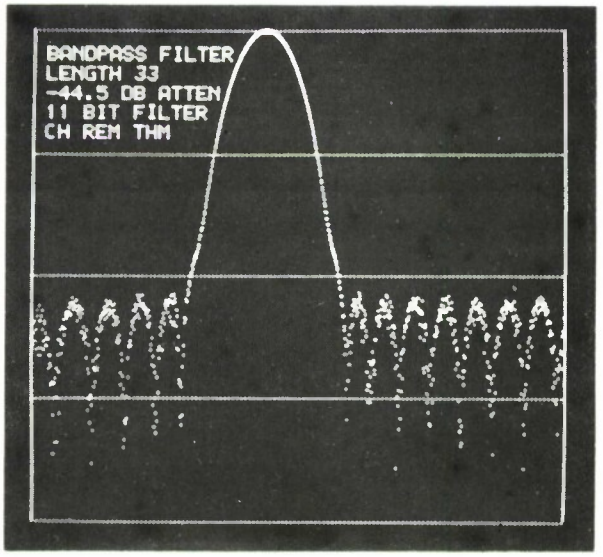


(b)

Fig. 13. Filter response with linear FN input using FNT convolution.
a) lowpass filter
b) bandpass filter



(a)



(b)

Fig. 14. Filter response with linear FN input using the FNT and the Chinese Remainder Theorem to increase precision.
a) lowpass filter
b) bandpass filter

X. Summary

A hardware implementation of the Fermat Number Transform has been built. In the course of designing this machine a new representation for numbers modulo $2^t + 1$ was derived to facilitate the arithmetic operations of the FNT butterfly. The design goal of a 40-MHz clock rate through the butterfly was nearly achieved with the final reliable clock rate being 38 MHz.

The FNT system was built as a peripheral device for the FDP and the software necessary use the FNT for convolution has been developed and was explained here.

Finally, a comparison with the FFT for special purpose pipeline hardware convolution has been made based on the present design. A conclusion of this comparison is that the FNT is a useful alternative if the data to be filtered are real and the computational elements are a large part of the convolver cost as in a pipeline architecture.

REFERENCES

1. J. M. Pollard, "The Fast Fourier Transform in a Finite Field," Math. Comp. 25, 365-374 (1971).

2. C. M. Rader, "Discrete Convolution via Mersenne Transforms," IEEE Trans. Computers C-21, No. 12, 1269-1273 (1972), DDC AD-7589380.

3. R.C. Agarwal and C.S. Burrus, "Fast Convolution using Fermat Number Transforms with Applications to Digital Filtering," IEEE Trans. Acoustics, Speech, and Signal Processing ASSP-22, No. 2, 87-97 (1974).

4. R.C. Agarwal and C.S. Burrus, "Fast One-Dimensional Digital Convolution by Multi-Dimensional Techniques," IEEE Trans. Acoustics, Speech, and Signal Processing ASSP-22, 1-10 (1974).

5. R. C. Agarwal and C.S. Burrus, "Number Theoretic Transforms to Implement Fast Digital Convolution," Proc. IEEE 63, (1975).

6. R. J. Purdy, et al., "Digital Signal Processor Designs for Radar Applications," Lincoln Laboratory Technical Note 1974-58 (December 1974), Vol. 1, DDC AD-B001419L and Vol. 2, DDC AD-B001420L.

7. L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing (Prentice-Hall, Englewood Cliffs, New Jersey, 1974).

8. P. E. Blankenship and E. M. Hofstetter, "Digital Pulse Compression via Fast Convolution," IEEE Trans. Acoustics, Speech, and Signal Processing ASSP-23, No. 2, 189-201 (1975).

9. J. W. Cooley, P.A.W. Lewis, and P.D. Welch, "The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine, and LaPlace Transforms," J. Sound Vib. 12, 315-337 (1970).

10. B. Gold and T. Bially, "Parallelism in Fast Fourier Transform Hardware," IEEE Trans. Audio Electroacoust. AU-21 No. 1, 5-16 (1973), DDC AD-7635730.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-75-149	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Hardware for the Fermat Number Transform		5. TYPE OF REPORT & PERIOD COVERED Technical Note 6. PERFORMING ORG. REPORT NUMBER Technical Note 1975-18
7. AUTHOR*(s)* McClellan, James H.		8. CONTRACT OR GRANT NUMBER*(s)* F19628-73-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 7X263304D215
11. CONTROLLING OFFICE NAME AND ADDRESS Ballistic Missile Defense Program Office Department of the Army 1320 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE 1 April 1975 13. NUMBER OF PAGES 62
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB Bedford, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES None		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* FNT FFT FDP peripheral pipeline convolvers direct form convolvers multipliers filter implementation arithmetic logic design		
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)* The design and implementation of a hardware Fermat Number Transform (FNT) is described. The arithmetic logic design is treated in detail and a new data representation for integers modulo a Fermat number is derived. Some results of filter implementation with the FNT are shown to illustrate the use of the hardware. Finally, the FNT is compared with the Fast Fourier Transform (FFT) on the basis of hardware required for a pipeline convolver.		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*